2010 12th International Workshop on Cellular Nanoscale Networks and their Applications (CNNA)

Approximating Euclidean Distance Transform with Simple Operations in Cellular Processor Arrays

Samad Razmjooei and Piotr Dudek
School of Electrical and Electronic Engineering
The University of Manchester
Manchester, UK
Email: samad.razmjooei@postgrad.manchester.ac.uk, p.dudek@manchester.ac.uk

***Abstract*—This paper presents a new algorithm for computing a distance transform, particularly suitable for massively parallel cellular processor arrays. The proposed Enhanced City Block Distance Transform (ECBDT) achieves good approximation to Euclidean distances, operating with 'increment' and 'minimum' operations only, and requiring only local 4-neighbour communication. The distance values are calculated in a wave-propagating manner, and are suitable for implementation on asynchronous processor arrays. The performance of the algorithm is adjustable through parameters. Presented simulation results illustrate the operation of the algorithm, and discuss the accuracy of the distance approximation that is achieved in comparison to Euclidean, City Block, Chessboard and Chamfer distance transforms.**

I. INTRODUCTION

In digital image processing, a map of distances is often required in which all non-feature pixels are assigned distance values to their nearest feature pixels. The map is called the Distance Transform (DT). The exact distance between two points with coordinates of (x_1, y_1) and (x_2, y_2) can be calculated from the Euclidean Distance formula:

$$D_{Eu} = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2} \quad (1)$$

Calculating the Euclidean Distance Transform (EDT) is a global operation, as for each non-feature pixel a nearest feature pixel must be found and a distance calculated according to (1). Therefore the algorithms to generate such distance map are computationally intensive. Furthermore, calculation of distances in presence of obstacles is problematic. To overcome these problems, other distance metrics such as City Block (Manhattan), Chessboard or Chamfer have been proposed to calculate the distance considering only small neighbourhoods [1]-[2]. The transforms that apply these metrics could be considered as giving an approximation to Euclidean distance with considerably less complex algorithms [3]. In particular, they are suitable for parallel implementation on cellular processor arrays, such as [4]-[8].

In the parallel DT algorithms, global distances are calculated by 'propagating' local distance values. Among the proposed distance transforms algorithms, City Block Distance Transform (CBDT) is the simplest and fastest to be implemented. It requires only 'increment' and 'minimum' operations on a 4-connected neighbourhood. However, the obtained distances differ largely from Euclidean distances. This may be undesirable in many applications. Chessboard DT is similar in this respect. Chamfer DT is a better approximation to Euclidean distance, but it requires a greater number of more complex operations (including multiplication and usually larger neighbourhoods) to calculate the distances [1]-[2].

In this paper, a parallel DT algorithm is proposed that retains the simplicity of CBDT, but gives a considerably better approximation to EDT. The algorithm can be executed iteratively on synchronous Single Instruction Multiple Data (SIMD) fine-grain processor arrays; it can also take advantage of asynchronous wave-propagating operations available on some cellular processor array hardware [4].

In the next section, the parallel implementation of CBDT is overviewed. In Section III, the proposed Enhanced City Block DT (ECBDT) is introduced and the methods of evaluating the results are presented. The experimental results are discussed in Section IV and conclusions are in Section V. The algorithm is designed for massively parallel cellular processor arrays, however the experiments and simulations presented in this paper have been conducted in MATLAB.

II. CITY BLOCK DISTANCE TRANSFORM

The CBDT measures the distance between the pixels based on the four direct neighbours (North, East, West, and South). The local distance value between a pixel and its direct neighbour is equal to one. The City Block distance between pixels with coordinates of (x_1, y_1) and (x_2, y_2) is calculated by:

$$D_{CB} = | x_2 - x_1 | + | y_2 - y_1 | \quad (2)$$

To calculate the CBDT of a binary image, initially the values of all the non-feature pixels are changed to infinity (or a suitably large number) and the feature pixels are set to zero [1]. The distance map is derived through iterations. In each iteration the distance values of direct neighbours are incremented by one:

$$P^n_{x,y} = \min(P^{n-1}_{x,y},\ P^{n-1}_{x,y-1} + 1,\ P^{n-1}_{x,y+1} + 1,\ P^{n-1}_{x-1,y} + 1,\ P^{n-1}_{x+1,y} + 1) \quad (3)$$

where $P^n_{x,y}$ is the distance value of the pixel in position (x, y) at iteration n. This process is repeated until convergence. The

978-1-4244-6678-8/10/$26.00 ©2010 IEEE

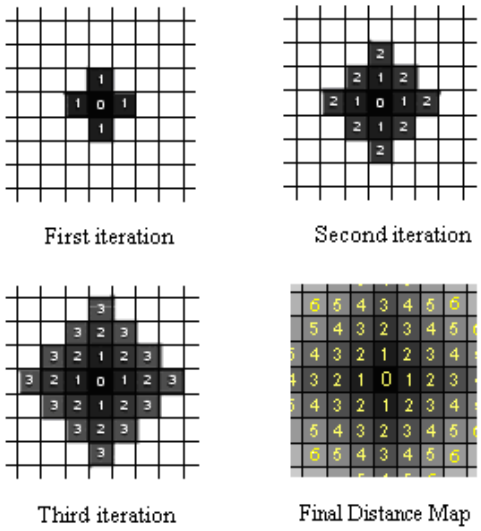


Figure 1. Propagation of local distances in distance map.

distance values are propagated from the boundary of the feature pixels to the neighbours until the distances for all the pixels are calculated. Fig. 1 illustrates an example of the propagation of CBDT for a binary image which consists of one feature pixel in the centre. The distance values in the DT matrix are shown using a gray-level scale.

It should be noted that the discrete-time iterations in (3) can be replaced by a continuous-time process, where the update equation is constantly evaluated. Processors such as ASPA [4] are capable of performing such operations and consequently the propagation of distances, as illustrated in Fig. 1, is asynchronous. The system always converges to the same solution, irrespective of propagation speed.

As compared with Euclidean distance, CBDT overestimates the distances for pixels located diagonally from their nearest feature pixels. This can be clearly seen in Fig. 2(b) as a distortion from the ideal isotropic distance map shown in Fig 2(a). Here we propose Enhanced City Block Distance Transform (ECBDT), which operates with "+1" and "minimum" operations only, similar to CBDT, but provides a much better approximation to the Euclidean DT (see Fig.2 (c)).

III. ENHANCED CITY BLOCK DISTANCE TRANSFORM

A. *Masked-Cells*

The *Masked-Cells* (MCs) in the calculation of a DT are defined here as cells that have their DT values equal to the minimum value of their direct neighbours. When calculating their modified CBDT, the MC update does not require an increment. Equations (4) and (5) are thus used to update the DT values in iteration n, depending on whether the pixel is a MC or not:

If (x, y) is not a MC:

$$P^{n}_{x,y} = \min(P^{n-1}_{x,y}, P^{n-1}_{x,y-1}+1, P^{n-1}_{x,y+1}+1, P^{n-1}_{x-1,y}+1, P^{n-1}_{x+1,y}+1) \qquad (4)$$

If (x, y) is a MC

$$P^{n}_{x,y} = \min(P^{n-1}_{x,y}, P^{n-1}_{x,y-1}, P^{n-1}_{x,y+1}, P^{n-1}_{x-1,y}, P^{n-1}_{x+1,y}) \qquad (5)$$

For example, consider the array shown in Fig. 3, with one feature pixel in the centre. Some of the cells are marked as MCs. Regular background pixels have their DT values incremented with respect to their smallest direct neighbours, while the DT values of MCs are equal to their smallest neighbours.

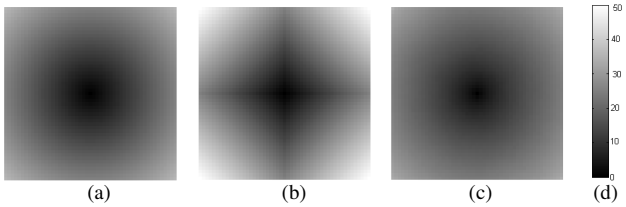


Figure 2. Distance Transforms of a binary image of 51x51 with one simple feature pixel in the centre. (a) Euclidean Distance. (b) CBDT. (c) ECBDT (Scaled). (d) The gray-level bar represents the distance values.

Given that the distance values are calculated in an iterative manner by propagating the local distances, a MC can impact all the cells that have their DT value depending on it. Fig. 4 illustrates how the DT result can be randomly distorted as the number and location of the MCs in the matrix is varied.

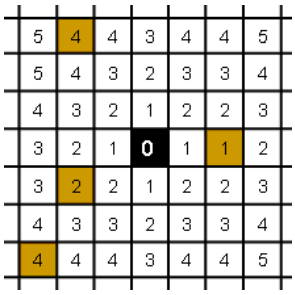


Figure 3. The effect of MCs on the DT. Four pixels are marked as MCs.

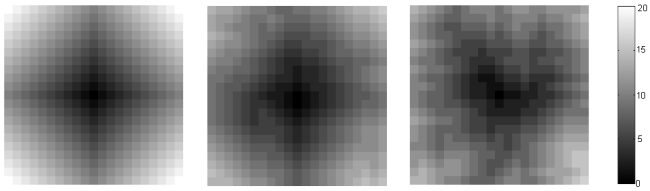


Figure 4. The effect of MC on matrix of 21x21 with one feature pixel in the centre. (a) CBDT with no MCs. (b) 10% of MCs. (c) 20% MCs; (d) The gray-level bar represents the distance values.

B. *ECBDT Algorithm*

To compute the ECBDT we use number of random patterns (k) of specific density of MCs (p), and apply the modified CBDT for each pattern, iterating equations (4) and (5) until

convergence. As random MC locations lead to random distortions of the modified CBDTs matrix for each pattern, we finally average the results to reduce the error.

The pseudo-code implementing this algorithm is shown in Fig. 5. Provided the processing hardware is capable of fast propagation of distance values, calculating multiple modified CBDTs can be easily performed, within a reasonable execution time.

Fig. 2 (c) illustrates the DT results when ECBDT has been performed for p=50% of MCs on k=200 randomly selected patterns. As it can be seen in Fig. 2, the result of ECBDT appears qualitatively closer to the EDT than the CBDT result. In the following sections we quantify this, and analyse the error with respect to ECBDT parameters such as MCs density and number of averages.

```
SumDT=0          // matrix accumulating the DT results
For 1 to k do    // k is number of averages
  G=Generate Random Pattern ( p)   //p is density of MCs
  D = DT(Input, G)      //Find the DT according to (4),(5)
  SumDT= SumDT + D      //Accumulate results
Next
ECBDT(k, p) = SumDT / k    //Calculate average
```

Figure 5. The pseudo-code to compute the ECBDT.

C. Error Calculation

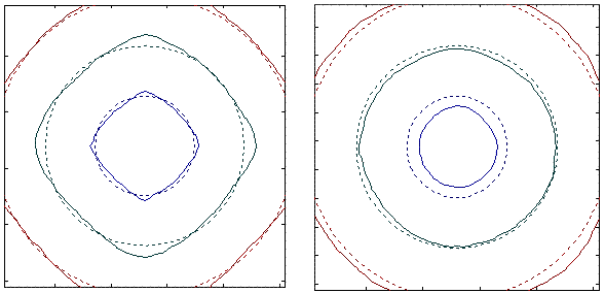
Two methods are introduced to calculate the error:

1) Absolute Error: To quantify the absolute error ε_{abs} between ECBDT and EDT, the RMS error is calculated across the DT matrix.

$$\varepsilon_{abs} = \sqrt{\frac{1}{n \times m} \sum_{x=1}^{n} \sum_{y=1}^{m} (\alpha_{abs} P_{x,y} - P'_{x,y})^2} \tag{6}$$

where n, m are number of rows and columns of the binary image respectively; $P'_{x,y}$ and $P_{x,y}$ are result of EDT and ECBDT, respectively, at position (x, y). Since the introduction of MCs leads to a systematic underestimate of the distance values in the DT matrix, an absolute scaling factor α_{abs} is introduced. The value of α_{abs} is defined as the optimal value that compensates the gain error in the DT calculation, i.e. the value that minimizes the absolute error (ε_{abs}) given by (6).

2) Relative Error: The absolute error gives a good indication of the overall difference between EDT and ECBDT. However in many applications the absolute error is less important than the uniformity of the DT result across orientations, i.e. a relative error between equidistant pixels (in a Euclidean sense) and the DT result. Fig. 6 illustrates two ECBDTs applied to a matrix, when two different sets of parameters have been used. Distances to a centre pixel are calculated. The resulting equidistant pixels for distances of 10, 20 and 30 (solid contours) and Euclidean equidistant pixels (dashed contours) are shown on both DTs for comparison. It can be seen that the absolute error of DT shown in Fig.6(a) is smaller than that of DT shown in Fig.6(b). However, distances in the horizontal direction are underestimated while those calculated diagonally are overestimated. On the other hand, the result in Fig.6(b) has a larger overall absolute error. However, it has a desirable property that equidistant pixels are forming contours much closer in shape to the ideal circles of EDT.



(a) (b)

Figure 6. Comparing the equidistant pixels between scaled ECBDT (solid) and EDT (dashed); Different ECBDT parameters are used in two cases: (a) k=500, p=15%, results in ε_{abs}=1.16; (b) k=500, p=50%, results in ε_{abs} = 1.31.

To quantify the relative error we use the following method. Distance transforms (ECBDT and EDT) are calculated for an image with a single feature pixel in the centre. A number (M) of concentric circles, centered on the feature pixel, are selected. In the experiments the radius of the circles is chosen with step s=5 pixels i.e. 5, 10, 15 etc (see Fig. 7). For each circle, a relative scaling factor α_r is calculated:

$$\alpha_r = \frac{\sum_{x,y \in C_r} P'_{x,y}}{\sum_{x,y \in C_r} P_{x,y}} \tag{7}$$

Where $(x,y) \in C_r$ are a set of N points lying on a circle C_r with radius $r \times s$, (see Fig. 7); $P_{x,y}$ and $P'_{x,y}$ are the calculated ECBDT and EDT values, respectively, at position (x, y).

For each circle, the variance σ_r^2 is then calculated according to (8) and finally the relative error ε_{rel} is calculated considering all circles, according to (9).

$$\sigma_r^2 = \frac{1}{N} \sum_{x,\, y \in C_r} (\alpha_r P_{x,y} - P'_{x,y})^2 \tag{8}$$

$$\varepsilon_{rel} = \sqrt{\frac{1}{M} \sum_{r=1}^{M} \sigma_r^2} \tag{9}$$

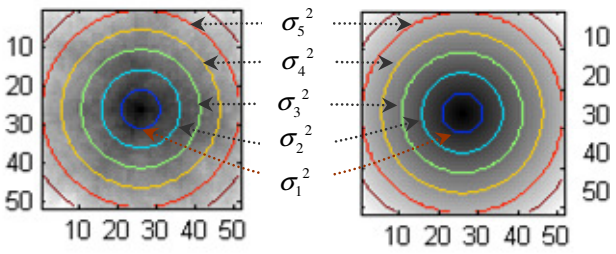


Figure 7. Calculating the variance. ECBDT shown on the left and EDT shown on the right. In this example, i.e. M=5, s=5 and r=1, 2, 3, 4, 5.

IV. Experimental Results

All numerical experiments reported in this section, have been performed on a binary image of 201x201 having one feature pixel in its centre. The ECBDT result depends on two parameters, the density of MCs, p, and the number of averages, k. Fig. 8 illustrates how the ECBDT result changes with these parameters. A large k results in smooth equidistant contours, while larger p improves their circularity. For smaller p, the contours have distorted shapes (resembling CBDT). They become more circular, but more noisy, as p is increased. The execution time of the algorithm will be directly proportional to k, and independent of p.

To investigate which p and k might be most optimal, the errors have been calculated by sweeping these parameters, as shown in Fig. 9. Since the result of ECBDT is non-deterministic, the plots in Fig. 9 are based on the mean values of the errors, averaged across many trials. In case of a small number of averages, the error values calculated using (6) and (9) exhibit large standard deviation, as shown in the example in Fig. 10. From the plot of ε_{abs} in Fig. 9 it can be seen that depending on k, about 20-25% of MCs results in lower absolute error and consequently gives a closer overall approximation to EDT. When we consider the relative error, it can be seen that

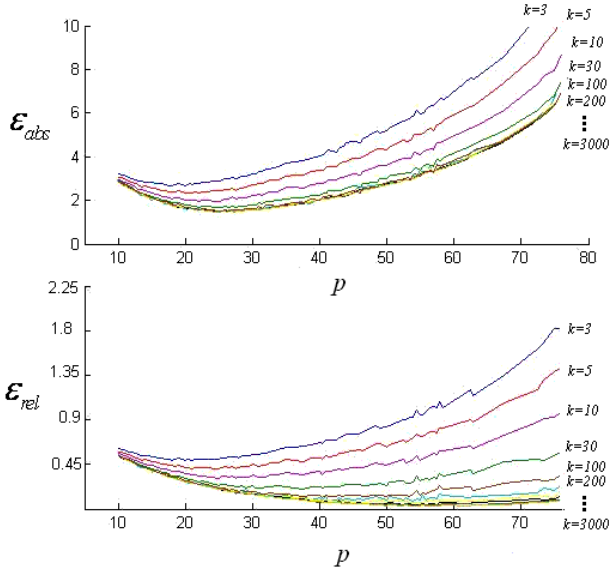


Figure 9. Absolute error (ε_{abs}) and relative error (ε_{rel}) vs. density of MC (p) for different number of averages (k).

the optimal density of MCs becomes larger as the number of averages increases. Fig. 9 can be used as a guide to select the appropriate parameters in practical applications.

Fig. 11 illustrates how the absolute scaling factor changes against the density of MCs. As we are increasing p the scaling factor increases.

The graphs in Fig. 12 show relative scaling factor against distance values in ECBDT, for different p. Averaging has been performed over k=500 random patterns. As the number of MCs is increased, the distances calculated using ECBDT become more nonlinear with respect to EDT. The nonlinearity of distance values in ECBDT can be observed in Fig. 6 (b); while the inner contour is smaller than its EDT counterpart, the outer contours is bigger. The plot in Fig. 12 can be used to compensate the non-linearity, for example using a look-up table, if required by the application.

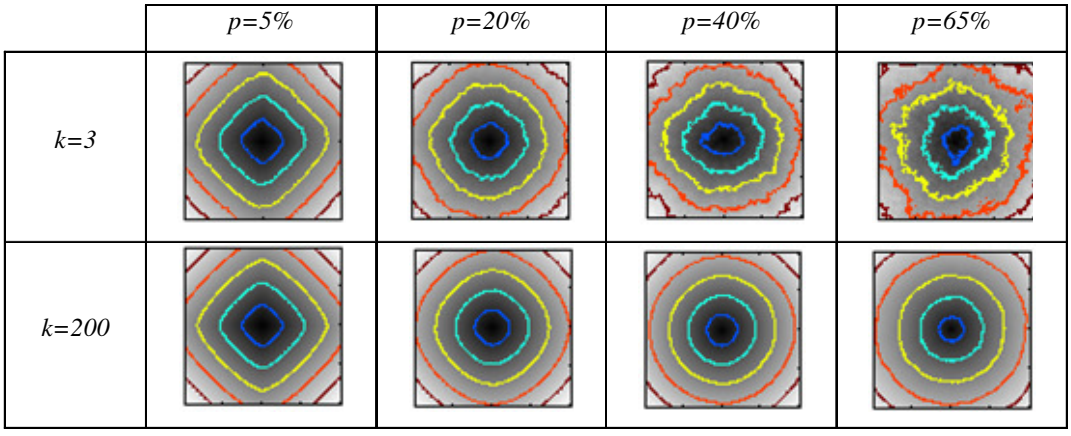


Figure 8. ECBDT result for different parameters. k is number of averages and p is density of MCs. Equidistant contours are shown. For small p the DT is similar to CBDT. As p is increased, it is possible to achive a DT very close to EDT in terms of circularity. Increasing k reduces the noise.

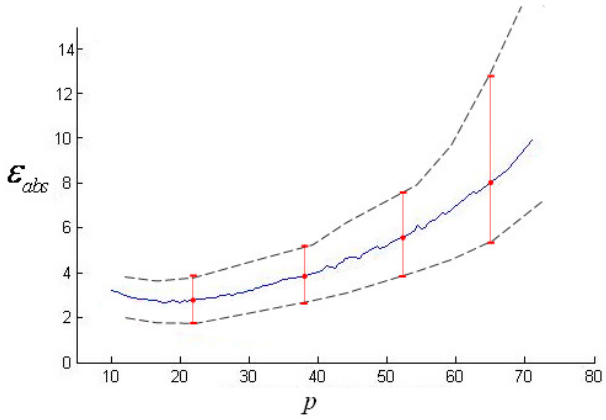


Figure 10. Absolutre error for small number of averages, k=3. Dashed lines follow error bars showing standard deviation of the value calculated using (6) across many runs of the ECBDT.

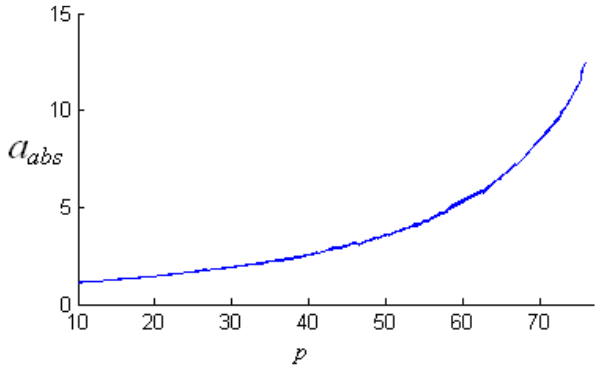


Figure 11. Absolute scaling factor (a_{abs}) vs. density of Masked-Cells (p).

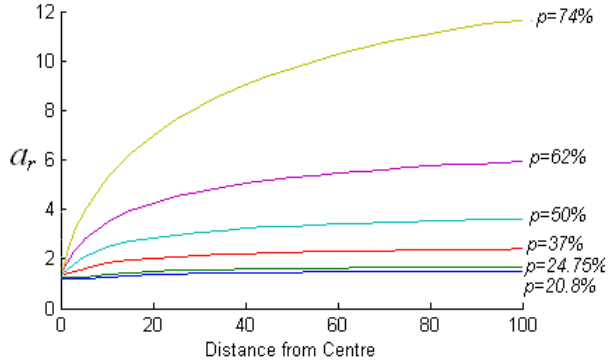


Figure 12. Relative scaling factor (a_r) vs. distance value from centre for different density of MCs

Table 1. Comparing the DTs to ECBDT

DT	ε_{abs}	ε_{rel}	Speed
City Block DT(CBDT)	27.7	6.0	Very fast
Chessboard DT	13.8	5.9	Very fast
Chamfer DT (3-4)	3.0	1.7	Fast
Chamfer DT (5-7-11)	0.8	0.6	Slow
ECBDT (k=5, p=20%)	2±0.5	0.46±0.2	Fast
ECBDT (k =50, p =35%)	1.8±0.2	0.25±0.1	Slow
ECBDT (k =100, p =40%)	2.1	0.18	Slow

V. COMPARISONS AND CONCLUSION

Table 1 shows a comparison of the performance of several DTs. The second and third columns compare the absolute error and the relative error for different types of DTs. The ECBDT gives a clear improvement in the quality of approximation to EDT, in comparison to CBDT and Chessboard DT. Chamfer DTs can give a better approximation but require more complex operations. The advantage of ECBDT is in that it can achieve a very good approximation to EDT using only a simple set of arithmetic operations on nearest neighbours and using limited hardware resources. In particular, all steps of the algorithm (generation of a random map, calculation of a modified CBDT, averaging of results) can be carried out using only a few memory elements per pixel, which makes the algorithm suitable for execution on pixel-level cellular processor arrays.

The ECBDT is flexible in the sense that depending on the application, the precision of the output DT and the speed of the algorithm can be defined by the density of MCs and number of averages. If an application requires speed, then approximately 20% of MCs and a low number of averages (e.g. k=3 or k=5) can be applied, with good results. If the relative distances are a matter of interest in an application, then a larger number of averages and MCs should be used.

The execution speed of ECBDT depends on the actual implementation. When implemented on cellular processor arrays which are capable of rapidly iterating equations (4) and (5), especially on processors that support asynchronous data propagations involving 'min' and '+1' operations, the ECBDT can provide a better performance than other methods of approximating Euclidean distance transform.

REFERENCES

[1] G. Borgefors, "Distance transform in digital image," Computer Vision, Graphics and Image Processing. 34, pp. 344-371, Feb. 1986

[2] A. Rosenfield and J. L. Pfaltz, "Distance function on digital pictures," Pattern Recognition, Pergamon Press 1968, Vol.1, pp. 33-61, Oct. 1967

[3] Per-Erik Danielsson, "Euclidean distance mapping," Computer Graphics and Image Processin 14,, pp. 227-248, Feb. 1980

[4] A. Lopich and P. Dudek, "ASPA: Focal Plane Digital Processor Array with Asynghronous Processing Capabilities," ISCAS 2008, pp 1592-1596, May 2008.

[5] P.Dudek and P.J.Hicks, "A General-Purpose Processor-per-Pixel Analog SIMD Vision Chip," IEEE Transactions on Circuits and Systems - I, vol. 52, no. 1, pp. 13-20, January 2005

[6] G. Linan, R. Dominguez-Castro, S. Espejo and A. Rodriguez-Vazquez, "ACE16k: a programmable focal plane vision processor with 128x128 resolution", ECCTD 2001, pp 345-348, August 2001.

[7] T. Komuro, S. Kagami, I. Ishii and M. Ishikawa, "Device and System Development of General Purpose Digital Vision Chip", Journal of Robotics and Mechatronics, Vol.12, No.5, pp.515-520 (2000)

[8] J. Flak, M. Laiho, A.Paasio, K. Halonen,"Dense CMOS implementation of a binary-programmable cellular neural network". Int. Journal of Circuit Theory and Applications, 2006. 34(4): p. 429-443.